Deformation Grammars: Hierarchical Constraint Preservation Under Deformation

Ulysse Vimont, Damien Rohmer, Antoine Begault, Marie-Paule Cani

▶ **To cite this version:**

Ulysse Vimont, Damien Rohmer, Antoine Begault, Marie-Paule Cani. Deformation Grammars: Hierarchical Constraint Preservation Under Deformation. Computer Graphics Forum, 2017, 36 (8), pp.429-443. 10.1111/cgf.13090 . hal-01518534

HAL Id: hal-01518534
https://inria.hal.science/hal-01518534

Submitted on 5 May 2017

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Volume XX (2017), Number XX pp. 1–14 COMPUTER GRAPHICS *forum*

Deformation Grammars: Hierarchical Constraint Preservation under Deformation

Ulysse Vimont [1], Damien Rohmer[1,2], Antoine Begault[1], and Marie-Paule Cani[1]

[1]Inria, Univ. Grenoble Alpes, Grenoble INP & CNRS (LJK), [2]CPE Lyon

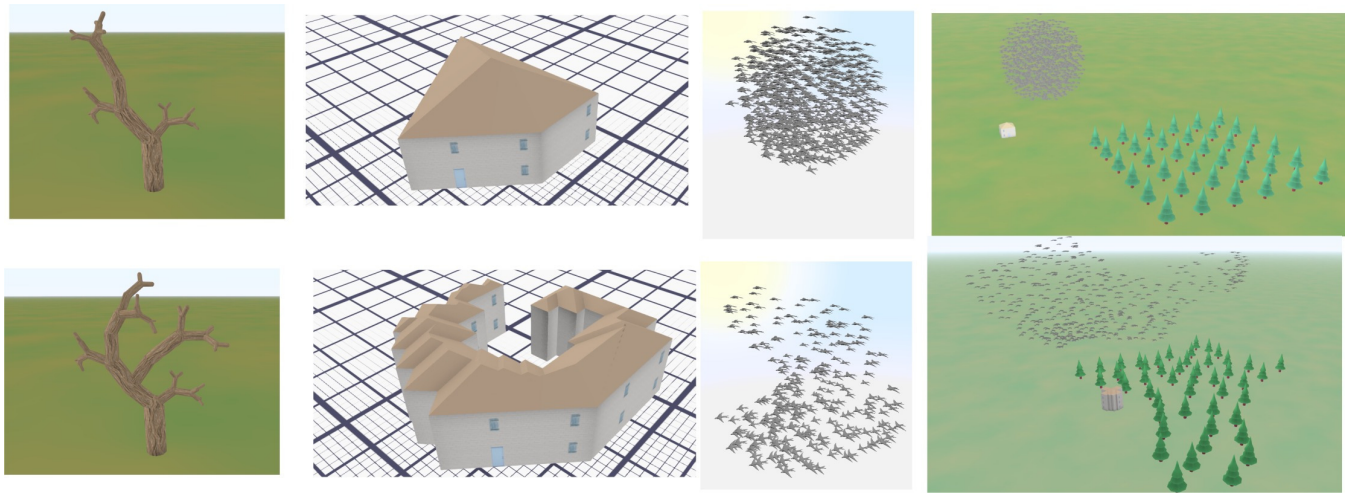


Figure 1: Deformation grammars allow to freely deform complex objects or object assemblies, while preserving their consistency. Top row: Original hierarchical objects (tree, house, bird flock, scene with mixed elements). The tree and the bird flock are made of parts of the same type, while the other objects are hetegogeneous hierarchies. Bottom row: Deformed objects, where the interpretation of user-controled deformations through deformation grammars is used to automatically maintain consistency constraints.

Abstract

Deformation grammars are a novel procedural framework enabling to sculpt hierarchical 3D models in an object-dependent manner. They process object deformations as symbols thanks to user-defined interpretation rules. We use them to define hierarchical deformation behaviors tailored for each model, and enabling any sculpting gesture to be interpreted as some adapted constraint-preserving deformation. A variety of object-specific constraints can be enforced using this framework, such as maintaining distributions of sub-parts, avoiding self-penetrations, or meeting semantic-based user-defined rules. The operations used to maintain constraints are kept transparent to the user, enabling them to focus on their design. We demonstrate the feasibility and the versatility of this approach on a variety of examples, implemented within an interactive sculpting system.

Categories and Subject Descriptors (according to ACM CCS): I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling—Hierarchy and geometric transformations

1. Introduction

Modeling and editing complex objects or shape assemblies is one of the bottlenecks of the virtual content production pipeline, despite a large body of work in the last two decades. While recent technical improvements enabled to model complex and detailed shapes, their creation and editing are still very tedious, and the quantity and quality of new 3D content produced fails at matching the increasing expectations of users.

Ideally, once a given 3D object or shape assembly is modeled, artists should be able to reuse it in different scenes after the ap-

© 2017 The Author(s)
Computer Graphics Forum © 2017 The Eurographics Association and John Wiley & Sons Ltd. Published by John Wiley & Sons Ltd.

propriate modifications. To achieve this, digital artists should be able to deform complex objects efficiently and in an intuitive way. For instance, stretching the model of a tree should either elongate branches or grow more of them depending on the user's intent, while insuring in both cases that the deformed object is still a valid model of a tree (e.g. branches are not intersecting).

This problem is made even more challenging when dealing with complex 3D models made of heterogeneous components. These models can be seen as a hierarchical organization of different sub-parts, each part (or sub-sets of parts) possibly needing to be deformed in a specific way. For instance, if the aforementioned tree model includes leaves and fruits, the latter may only be scaled in a uniform way, although branches can also be scaled along their main axis. Houses with doors, windows, and walls made of bricks, or animals with scales and appendices, are other examples of such hierarchical heterogeneous shapes. Dealing with the deformation of these complex models is currently tedious as each type of sub-part may require the use of a specific deformation tool in order to remain consistent. Moreover, deforming each part independently is not sufficient for maintaining the consistency of the whole: the user needs to manually ensure that inter-part dependencies are respected.

Our work addresses consistent deformation of complex objects. More precisely, we propose a unified framework to handle artist-driven deformation set-up for hierarchical heterogeneous objects. We claim that the following features are essential for an artist-driven deformation tool, and address them specifically:

- The validity, or consistency, of the whole model should be maintained throughout the deformation. The user should be able to select the consistency criteria for each type of element and at different scales, in order to fully express their intent.
- A hierarchical model should be editable at different scales, ranging from local to global ones.
- The artist should be able to apply the edits in the order they wish, not only from coarse to fine scales.

Our solution is based on the new concept of *deformation grammars*. The latter enable to define deformation interpretation rules and allow to freely deform a complex object while maintaining its consistency. Our contributions include formal definitions for the notions of complex object, deformations and consistency (Section 3), the description of a general formalism for *deformation grammars* (Section 4), and their extension enabling to freely interleave local and global editing (Section 5). We present a variety of applications and results produced within an interactive sculpting software (Section 6), and discuss the advantages and limitations of our deformation framework (Section 7) before concluding.

2. Previous Work

Being able to create and edit shapes in an intuitive way has been a major goal of computer graphics research for many years.

Interactive and Procedural methods are the most usual ways to create geometric content. While standard 3D mesh editing tools enable to interactively model any desired shape, they may require tremendous efforts in the case of complex objects, defined as a hierarchy of many interdependent parts. Procedural modeling techniques have been very efficient to design complex structured objects from a set of parameters. However, they are usually dedicated to a specific type of objects, such as cities and buildings modeling [Cit, LSWW11, SKK*14, EBP*12, IMAW15], or trees modeling [PBPS99, FP99, BPF*22]. These methods require unfortunately a large amount of parameter tuning and only provide indirect control - through trials and errors - on the results. Interactive approaches have been proposed to bring global control over objects defined procedurally, but each approach requires a dedicated system without always providing direct local and global control on the object [BBP13, KW11, LRBP12].

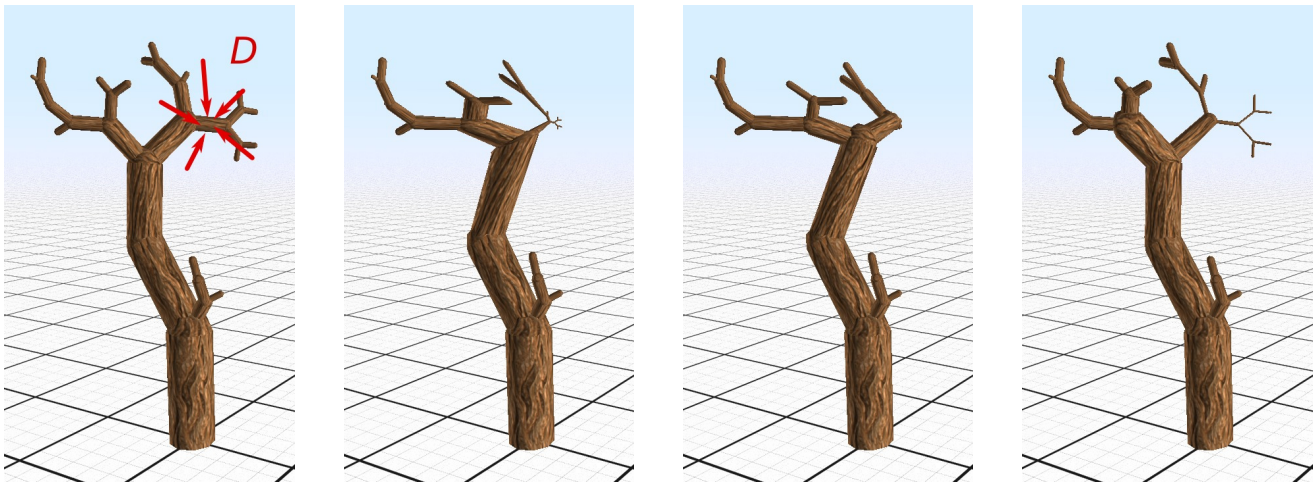
Standard deformation methods aim at deforming a mono-resolution shape while maintaining some of its properties. In the context of virtual sculpting of triangular meshes, this includes preserving volume while applying free-form deformations to a shape [ACWK04, vFTS06]; or enabling a mesh to maintain quasi-uniform sampling while undergoing free-form deformation that includes changes of topological genus [SCC11]. In Sorkine and Alexa's seminal work [SA07], the authors introduce a deformation scheme that preserves co-rotated distance vectors and tends to maintain local shape features under deformations.

In the above methods, the geometric property to be maintained during the deformation cannot be selected locally on the model. In consequence, objects are deformed as if they were made of some uniform elastic or plastic material. Moreover, the property which is preserved through deformation is not chosen by the user, and not fitted to every type of object.

Analyze-and-edit approaches use a two-step approach for deforming man-made objects in a way that maintains their structure [MWZ*13]. This means either preserving or duplicating specific details when the model is stretched. The first step aims at computing a set of features on an input model. In the second step, the identified features are automatically preserved while the user deforms the object. Features can be selected based on local geometric criteria such as saliency [DK14, ML13], curvature [KSSCO08], or wires [GSMCO09]; as well as based on higher-level properties such as linear arrangements [BWKS11, BWSK12], detail patterns [AZL12], element type adjacency matching [LVW*15], ergonomicity [ZLDM16], or 2D distributions of sub-shapes [EVC*15]. Yumer's work [YCHK15] allows for continuously deforming an input object through handles that represent semantic attributes. All these methods only consider two levels of detail (an object and its main parts or features), so their applicability to shape hierarchies is limited. More general hierarchy of deformations were also studied for generic meshes [GPCP13] but with limited consistency preservation.

Closer to our work, Zhen [ZFCO*11] computes specific controllers for the components of complex 3D models, allowing the user to deform the right degrees of freedom while maintaining inter- and intra-parts consistency. Controllers can be grouped, forming a hierarchy. Our method can be seen as a generalization of this work. We provide a unified framework for deforming hierarchical objects, enabling all the previous deformation modes to be

© 2017 The Author(s)
Computer Graphics Forum © 2017 The Eurographics Association and John Wiley & Sons Ltd.



(a) (b) (c) (d)

Figure 2: An initial complex object (a) is deformed by the user using a local tool applied at the top right. (b): The object is deformed without any deformation interpretation; Note how branches geometry is degenerated. (c): The deformation grammar allows us to hierarchically interpret the deformation in order to preserve the object consistency at each level of its hierarchy: branches remain cylindrical. (d): Changing the rules of the grammar, such as stating that branches can change radius but not length, allows us to change the deformation behavior while still respecting the object's consistency.

used, possibly at the same time, on different parts of the model or at different scales. Various deformation propagation schemes can be used and several alternative deformation modes can be specified for a given part. The task of defining and assigning the desired behavior, at the desired scale, to the components of a hierarchical model is left under the user's control, which allows for the creation of varied and expressive behaviors. Doing so, we skip the analysis step: Our method belongs to edit-only methods, described next.

Edit-only methods relax the *analyze-and-edit* paradigm, allowing the use of consistency criteria which would be very hard to analyze. For instance, Lipp [LWW08] enables the visual editing of shape grammar rules for modifying building appearance; Milliez [MWCS13] allows the mutation and duplication of object parts for structured shapes predefined using puzzle-grammars; Jordao [JPCC14] extend the previous method to crowd-patches embedding recomputed crowd animation data; Emilien [EPCV14] maintains the consistency of a waterfall scene under vector-based design; Longay [LRBP12] allows us to manipulate a realistic tree through sketching; and Stanculescu [SCCS13] extends quasi-uniform meshes [SCC11] for accounting for feature lines, using a taxonomy of possible behaviors.

Our method can be seen as a super-set of all these previous ones, allowing them to interoperate on different parts of an object. Since it also discards the analysis step, complex consistency criteria are allowed, leading to expressive deformation modes.

3. Hierarchical approach to object deformation

This section defines the notion of complex object, and formalizes the concepts of deformation and consistency on that type of object.

3.1. Complex objects

In the context of this work, we call *element*s any object of the scene. An element ε can be either a simple or a complex object:

- A *simple object* is a geometric object in the classical sense, fully defined by the set of *internal parameters* of its visual representation (such as a triangular mesh defined by the position of the vertices and their indexing into faces).
- A *complex object* is defined recursively as a set of internal parameters, plus a set of *hierarchical parameters* which are references to sub-objects. The sub-objects, are called the *children* of the element ε, and noted $C(\varepsilon)$. In return, an element is referred to as the *parent* of its *children*.

Note that the internal parameters of a complex object do not always correspond to a visual representation: For instance a heap of stones may include internal parameters such as slope or number of stones, while the visual representation may be only be stored in its children - e.g. simple objects representing the individual stones.

In our formalism, an element ε can only have a single *parent*, noted $p(\varepsilon)$. Pairs of elements $(\varepsilon_0, \varepsilon_1)$ are associated a *relation type* $t(\varepsilon_0, \varepsilon_1)$ that can be either *child*, *parent*, *self*(if $\varepsilon_0 = \varepsilon_1$) or *none* in all the other cases.

Lastly, a *semantic type* $t(\varepsilon)$ is associated with each element ε : for instance, the fact that the internal mesh represents a stone. A complex object with children of the same semantic type (such as the heap of stones we already described) is called *homogeneous*, otherwise it is called *heterogeneous*.

Example: Let us consider a naive model of a tree, where cylindrical trunk and branches subdivide at their extremity into a few smaller branches, as can be seen in figure 2a. This tree can be represented as

© 2017 The Author(s)
Computer Graphics Forum © 2017 The Eurographics Association and John Wiley & Sons Ltd.

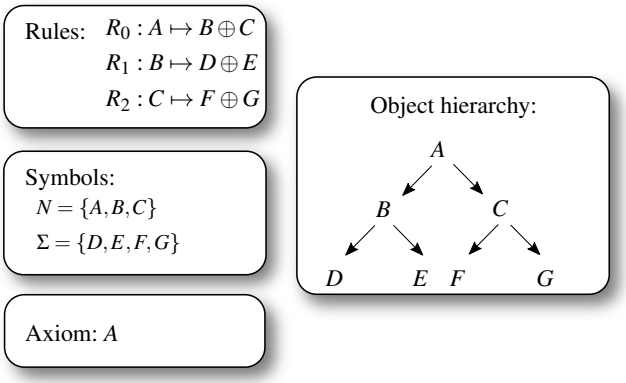


Figure 3: Standard use of grammars to define hierarchical shapes.

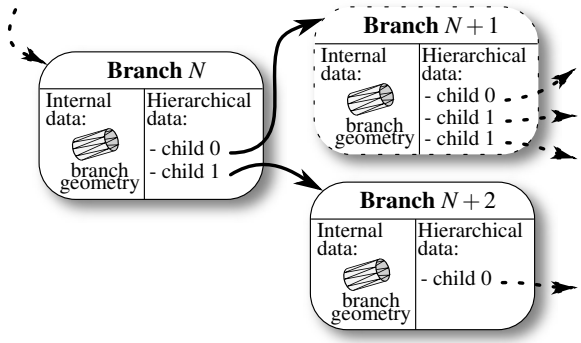


Figure 4: A tree model can be organized hierarchically into a complex object. This allows to consider individually each level of the hierarchy as well as the relationships between levels.

a complex object ε_0, using a large cylinder (for the trunk) as internal representation – in this case this representation is also visual – plus a set of references to main branches. Each branch is itself represented as a cylinder plus a set of references to children branches (see Figure 4). The smallest branches at the extremities are simple objects, with only a visual representation but no sub-branch. The other ones are complex objects. Both are of the same semantic type as ε_0.Therefore, this object is a homogeneous object.

The hierarchy of a complex object can be described manually by a user or may result from the use of a procedural modeling tool to build the object, such as a shape grammar or a L-system for a tree [PL12] (see figure 3, top-left). Alternatively, this hierarchy could be retrieved from an input shape using hierarchical segmentation [AFS06].

3.2. Deformation

A *deformation* is any function which maps the values of an object's parameters. For example, if the object is a mesh parameterized by a list of the vertices and their arrangement into faces, an example of deformation is a space deformation $d : \mathbb{R}^3 \to \mathbb{R}^3$ used to change the vertices positions. Classically, applying a deformation d to a complex object G (which we call *object deformation* and noted using the couple $D = (G, d)$) is performed by applying the deformation independently to the visual representations of each of the object's sub-parts. In contrast, our formalism enables us to redefine the application of a deformation in a hierarchical way, a first step for enabling us to preserve the consistency of complex objects through deformations. The hierarchical decomposition is done as follows:

An *element deformation* is defined as the edit of the element's internal parameters, which we call the *internal deformation*, followed by the element deformations applied to its children (if any), which we call *hierarchical deformation*. Using the element formalism for G, where $G = \varepsilon_0$ is the highest element in the hierarchy, this enables us to rewrite the object deformation of G in a hierarchical way, as follows:

$$\begin{cases} D^0 = (\varepsilon_0, d) \\ D^i = (\varepsilon_i, d) = D^i_{internal} \oplus D^i_{hierarchical} & \forall \varepsilon_i \in E \\ D^i_{hierarchical} = \bigoplus_{\varepsilon_j \in C(\varepsilon_i)} D^j \end{cases} \quad (1)$$

where $\oplus$ stands for the *independent application* of element deformations on a set of elements. This operator also allows us to combine any internal element deformation with other deformations.

The hierarchical definition of deformations in Equation 1 is instrumental for allowing various sets of consistency constraints to be maintained when deforming complex objects. Section 4 explains how we express this deformation propagation process using deformation grammars, and use it for preserving the consistency of the object.

3.3. Consistency

We call *element consistency* the set of properties that an element must satisfy to be considered as valid. Following our hierarchical approach, this notion is split into two sub-concepts: *internal consistency* and *hierarchical consistency*. Internal consistency of an element is based on its internal parameters: for example its length or curvature; It is therefore independent of any other element. Hierarchical consistency of an element relies the relation the element needs to maintain with its children, for example their relative positions or matching types.

We call *object consistency* the composition of the consistencies of all the elements composing an object, i.e. the set of all internal and hierarchical consistencies of the elements of the object.

Example: In the case of the simple tree model already discussed, the internal consistency of a branch could be for instance the cylindrical aspect of the internal triangular mesh that represents it. The hierarchical consistency can be the fact that the sub-branches all depart from the extremity of their parent branch, in addition to the fact that sub-branches are themselves consistent.

4. Deformation Grammars

This section introduces deformation grammars as an efficient tool to setup consistency-preserving deformations for complex objects.

© 2017 The Author(s)
Computer Graphics Forum © 2017 The Eurographics Association and John Wiley & Sons Ltd.

4.1. Definition

Formal grammars are widely used in Computer Graphics for representing hierarchical processes. More specifically, shape grammars such as the one in Figure 3 are often used to generate the static geometry of complex objects [MWH*06, EBP*12, SM15]. In this work, we extend the scope of formal grammars to handle the hierarchical deformation of complex objects.

A *deformation grammar* models a deformation behavior for a complex object under a set of deformations. It is defined as any formal grammar by:

- a set of non terminal symbols N.
- a set of terminal symbols Σ
- an axiom $A \in N$
- a set of production rules $P = \{R_i\}$

Symbols. A symbol is an element deformation $D = (\varepsilon, d)$, where $\varepsilon \in E$ is the *target* of the arbitrary deformation d. A terminal symbol is the deformation of the internal parameters of an element, while a non-terminal symbol requires further interpretation. Symbols can be assembled using the independent application operator $\oplus$.

Symbols inherit the partial ordering of the elements: $D_1 = (\varepsilon_1, d_1)$ is said to be *lower* than $D_2 = (\varepsilon_2, d_2)$ (noted $D1 < D_2$) if ε_2 is an ancestor of ε_1 in the object's hierarchy.

Besides, symbols are typed according to the types of both the target and the deformation:

$$t(\varepsilon, d) = (t(\varepsilon), t(d)) \tag{2}$$

Axiom. The axiom is a non terminal symbol created by the user. Its target is the uppermost element of the hierarchy. It represents the object deformation intent, such as a free-form deformation interactively generated through a sculpting tool. It is the initial symbol which will be decomposed into other symbols, until only terminal symbols remain.

Example: In our example, the axiom is the deformation that the user wants to apply to the tree model. It is processed as a deformation of the highest element ε_0 in the object's hierarchy (i.e. the trunk, see Equation (1)) and decomposed hierarchically using the grammar rules.

Rules. A rule is the substitution of a symbol by a $\oplus$ of other symbols. In other words, it is the translation of an element deformation into the deformations of its components. It is defined with respect to a type of symbol:

$$R(t(D)) : D \mapsto D' \tag{3}$$

where D' is an element deformation preserving the consistency of elements of the type $t(\varepsilon)$. Following Equation (1), we define D' as follows:

$$D' = D_{internal} \oplus D_{hierarchical} \,. \tag{4}$$

$D_{internal} = (\varepsilon, d')$ is a terminal symbol; It is a deformation that applies to the internal parameters of the element ε and preserve its internal consistency. $D_{hierarchical}$ is a non-terminal symbol; It calls for the independent application of deformations of the children of ε that preserve the hierarchical consistency of ε:

$$D_{hierarchical} = \bigoplus_{e \in C(\varepsilon)} D^{\varepsilon,e}(e) \,, \tag{5}$$

where $D^{\varepsilon,e} = (e, d_e)$ is an element deformation that preserves the hierarchical consistency between ε and e. It will be further processed for element e by the rule $R(t(e), t(d_e))$ for preserving the consistency of e (see Equation (3)).

The rules, in the form of Equation (3), are defined by the user for each type of symbol, by specifying $D^{\varepsilon}_{internal}(\varepsilon)$ and $D^{\varepsilon,e}$ used in Equations (4) and (5). They enable to control the behavior of a complex object under arbitrary deformations, and in particular to preserve the consistency of the object, as illustrated next.

4.2. Example

Let us come back to our example of the tree model and detail the process of creating a deformation grammar. The structure of the object has been described in section 3.1 and the associated consistency constraints in section 3.3.

Let the user apply a free form space deformation $d : \mathbb{R}^3 \to \mathbb{R}^3$ as deformation intent. The corresponding axiom is $A = (\varepsilon_0, d)$. The naive deformation of the element's internal parameters does not preserve the consistency of the object, as can be seen in Figure 2b.

A consistency preserving deformation can be set by using rules (3) and Equations (4) and (5) while defining:

- $D_{internal} = (\varepsilon, d')$ where d' is the affine transformation that moves the two extremities c^{ε}_0 and c^{ε}_1 of a cylinder to their images $d(c^{\varepsilon}_0)$ and $d(c^{\varepsilon}_1)$ respectively;
- $D^{\varepsilon,e} = (e, d)$. This is used to apply the input space deformation to e, which does not disconnect it from ε.

Using those definitions, applying the grammar rules preserves the cylindricity of the branches (internal consistency) as well as the adjacency between a branch and its sub-branches (hierarchical consistency), as shown in Figure 2c. Figure 2d shows that different rules can also preserve the object consistency while offering another deformation behavior.

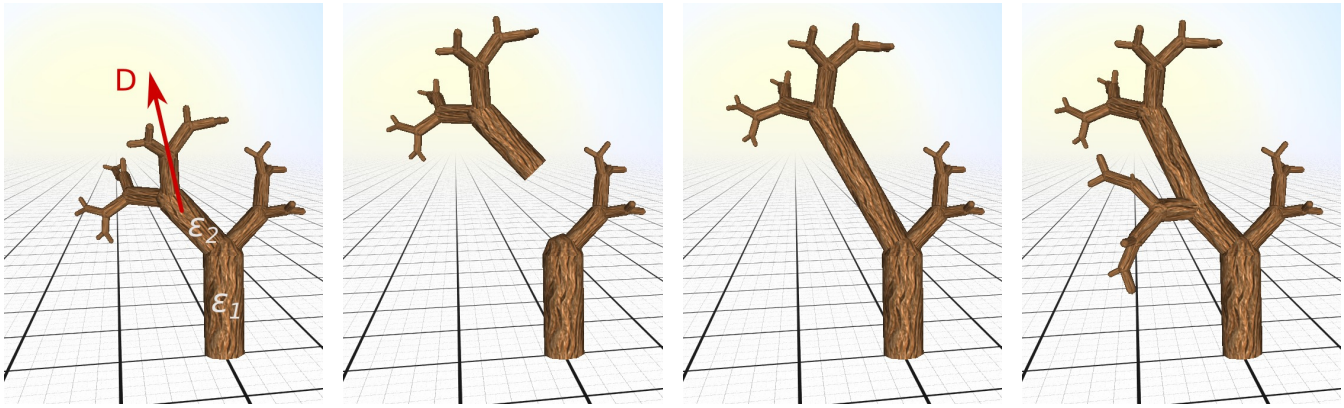
Appendix A fully describes the deformation grammar corresponding to this example.

5. Bilateral Grammar Rules

As stated in Section 1, an object should be editable at different scales (i.e. by editing parts at different levels of its hierarchy) in an arbitrary order. But the deformation grammars defined so far start from a deformation of the top level element ε_0 and propagates down to preserve the consistency. Figure 5b shows that applying the deformation on another element than ε_0 breaks the object consistency at the parent level. This comes from the parent rule not interpreting the deformation. This reduces a lot the amount of user control.

In order to maintain the object's global consistency during the deformation at any hierarchical level, deformation grammars needs

© 2017 The Author(s)
Computer Graphics Forum © 2017 The Eurographics Association and John Wiley & Sons Ltd.



(a) (b) (c) (d)

Figure 5: Bilateral grammar rules allow to deform a complex object at any level of the hierarchy. (a): An initial model has one of its elements deformed. (b): Result when a local edit is applied to a sub-branch without the use of bilateral grammar rules. (c): Using our bilateral grammar rules maintains consistency, here by ensuring that the edited branch stays in contact with its parent. (d): An alternative rule is used to automatically split elongated branches and generate new sub-branches.

to handle upward consistency management. This section presents bilateral grammars as an extension of descending grammars (presented in Section 4) allowing to solve this problem.

5.1. Closed-loop problem

Let us consider a tree model with two branches ε_1 and ε_2 such that $\varepsilon_2 \in C(\varepsilon_1)$. Now, ε_2 receives a deformation d from the user.

A naive solution to allow for upward consistency management consists in creating an *ascending grammar rule* $R(t(\varepsilon_2),t(d))$ generating a deformation of ε_1. For example, we could have $R_{ascending}(t(\varepsilon_2),t(d)) : D = (\varepsilon_2,d) \mapsto D' = (\varepsilon_1,d)$, where D' is a non terminal symbol. But D' would be translated into a deformation of ε_2 following Equation 5. This results in a loop creating non-terminal symbols, and never converging to terminal ones. The deformation operation does not terminate in this case.

5.2. Symbol specification

Our solution to the closed loop problem requires two additions to the symbols definition: a *source* and a *direction*. Rules can make use of this new data to avoid the closed-loop problem, as explained in the next Section (5.3).

A bilateral symbol is defined as:

$$D = (\varepsilon, d, S, \delta) \in \Sigma \cup N \tag{6}$$

where $\varepsilon \in E$ is the interpreting element, d the deformation, $S \in E$ is the original target of the deformation that we call the source, and $\delta = \{\uparrow,\downarrow\}$ is the direction of the symbol (ascending or descending).

The distinction between the *target* of the deformation and the element interpreting it allows to manage higher-than-source consistencies while keeping track of the element to deform. An ascending symbol indicates a deformation needing higher-level interpretation. Therefore, the axiom is set to have $S = \varepsilon$ and $\delta = \uparrow$.

The type of bilateral symbols account for its direction as well as for the types of its component (interpreting and target element, deformation) and for the relationship between the target and the interpreting element.

The next section describes how a symbol source and direction are accounted for in the rules.

5.3. Bilateral grammar rules

Depending on the object we consider, a symbol might need to be translated to the parent, the grand-parent, or any other ancestor of the target element, up to the highest element of the hierarchy. This is achieved by the generic ascending rule:

$$R_{up} : (\varepsilon, d, S, \uparrow) \mapsto (p(\varepsilon), d, S, \uparrow) \tag{7}$$

This rule translates the symbol by changing the interpreting element (setting it to be the parent of the current one) without modifying the other parameters of the symbol.

The appropriate level for interpreting the deformation depends on the object, and is characterized by the user using bilateral symbol types. Once R_{up} creates a symbol with the appropriate type, another rule will stop the ascent:

$$R_{interp} : (\varepsilon, d, S, \uparrow) \mapsto (C(\varepsilon, S), d_{interp}, S, \downarrow) \oplus K \tag{8}$$

where $C(\varepsilon, S)$ is the child of ε which is an ancestor of S, and K is a $\oplus$ of descending symbol targetted at the children of ε except $C(\varepsilon, S)$, and which deformation aims at preserving the hierarchical consistency of ε. Unlike R_{up}, R_{interp} is object-specific.

The descent takes place in two phases: The first one concerns symbols which interpreting element is higher than the source; The second one concern symbols which interpreting element is lower than the source, and is identical to the regular interpretation of a mono-lateral grammar.

© 2017 The Author(s)
Computer Graphics Forum © 2017 The Eurographics Association and John Wiley & Sons Ltd.

The first phase of the descent is achieved using the generic descending rule:

$$R_{down} : (\varepsilon, d, S, \downarrow) \mapsto (C(\varepsilon, S), d_{interp}, S, \downarrow) \quad (9)$$

Once reached the level where $\varepsilon = S$, the symbol is interpreted down as with a descending grammar. Any rule $R_{desc} : (\varepsilon_i, d_i) \rightarrow (\varepsilon_j, d_j)$ of a descending grammar can be formulated into a bilateral rule $R_{bi} : (\varepsilon_i, d_i, S, \downarrow) \rightarrow (\varepsilon_j, d_j, S, \downarrow)$

In order to prevent any interpretation loop, we state the following descending monotony principle: A descending symbol shall never be translated into an ascending one.

5.4. Example

Let us now extend the example of Section 4.2 in order to allow for bilateral deformation interpretation. We consider an affine deformation d applied on a branch $\varepsilon_2 \neq \varepsilon_0$ of a tree. According to Section 5.2, the axiom is $A = (\varepsilon_2, d, \varepsilon_2, \uparrow)$. It is processed by the rules R_{up} into $A_1 = (p(\varepsilon_2), d, \varepsilon_2, \uparrow)$.

We define R_{interp} such that:

$$R_{interp} : (p(\varepsilon_2), d, \varepsilon_2, \uparrow) \mapsto (\varepsilon_2, d', \varepsilon_2, \downarrow) \quad (10)$$

We define d' as the affine transformation which displaces one extremity of S while preserving the other:

$$\begin{cases} d'(p^{\varepsilon_2}_{start}) = d(p^{\varepsilon_2}_{start}) \\ d'(p^{\varepsilon_2}_{end}) = p^{\varepsilon_2}_{end} \end{cases} \quad (11)$$

On Figure 5c, we can see that the interpreted deformation keeps ε_1 and ε_2 connected, which respects the consistency of the tree. Figure 5d shows an alternative rule which splits elongated branches and generates new sub-branches. For that, we define R_{interp} as:

$$R_{interp} : (p(\varepsilon_2), d, \varepsilon_2, \uparrow) \mapsto (\varepsilon_2, d', \varepsilon_2, \downarrow) \oplus (\varepsilon_2, split, \varepsilon_2, \downarrow) \quad (12)$$

where *split* is a splitting deformation which occurs whenever a branch is too long.

The deformation grammar for this example is fully described in Appendix B.

5.5. Persistent editing

Enabling to apply deformations at different levels of the hierarchy greatly increases the user's freedom. With this method, small scale edits may, however, be overwritten by subsequent higher level modification, leading to the loss of specific user changes.

Bilateral deformation grammars allow to seamlessly solve this issue by keeping track of previously locally edited elements. Once an element ε is locally edited by the user, it can be tagged as *persistent*. Grammar rules can then take this tag into account for preserving such elements.

Therefore any global deformation applied later on the object will not modify the previously edited element enabling the user to iterate between global and local deformations as desired.

6. Applications and results

This section develops several possible applications, inspired by state of the art deformation methods, to demonstrate the versatility of our framework. All the examples were implemented using different deformation grammar rules, within the same interactive sculpting software. We also refer the reader to the video accompanying this work.

6.1. Grammar creation

Whatever the category of complex object to be deformed, the creation of a new deformation grammar proceeds as follows:

1. Design the hierarchy of the object, or use the hierarchy inherited from a previous procedural generation method;
2. For each type of element in the object, identify its internal and hierarchical consistency rules;
3. Based on steps 1 and 2, identify the deformation types applicable to each type of element;
4. Create a set of downward rules for each pair of element types and associated deformation type;
5. Optionally enrich the set of rules with upward rules allowing to maintain upward hierarchical consistency.

Since a rule is needed for every tuple $(t(element), t(deformation), t(relation))$, the number of rules to design is directly correlated to: the number of element types; for each element type, the number of possible deformation types; for each element type, the number of possible children types.

The example grammars presented in this section contain between two and ten rules.

6.2. Implementation

Results using deformation grammars were implemented as a C++ sculpting software. Details about the implementation are given in Appendix C, including an example UML class diagram in Figure 13.

Each element type corresponds to a class; Each class implements a `process()` function per deformation type it can handle. Grammar rules are implemented inside these functions, and the symbol to be processed is passed as argument.

A `process()` function can be called either from the event manager (in which case the source of the symbol is 0), or from another `process()` function (e.g. issuing from the element's parent during hierarchical interpretation).

Note that unlike shape grammars, deformation grammar symbols should be translated at interactive rates inside the sculpting framework. C++ hard-coded grammar rules are compiled along with the sculpting framework, which allows for fast symbol translation. Finally, rules are parameterized at run time using state variables enabling to switch of deformation behavior at run time. This is for example used for enabling/disabling the grammar interpretation in the examples illustrated in Figures 2 and 5.

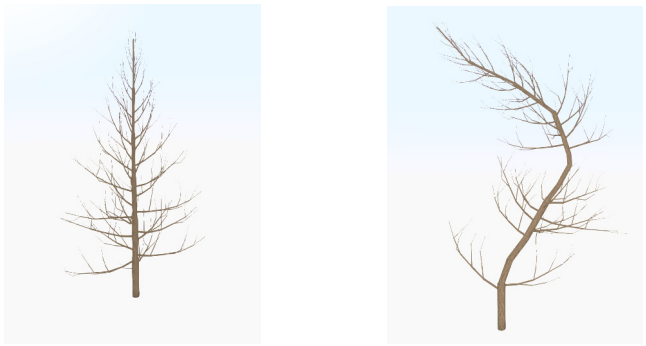
© 2017 The Author(s)
Computer Graphics Forum © 2017 The Eurographics Association and John Wiley & Sons Ltd.

Spatial complexity Recurrent function calls have a memory cost which increases linearly with the depth of recursion. Let us consider a complex object described by a hierarchy of depth d. The worst case scenario of recurrent call depth -i.e. the deepest element of the hierarchy calls the highest one through successive up-ward rules - entails at most $2d$ recursive function calls.

6.3. Organic shapes

We start by demonstrating our deformation grammar on complex objects representing organic shapes.

Figure 6 shows three steps of an interactive tree modeling session. In this example, we use the rules given in Section 4.2 to ensure that branches remain cylindrical and adjacent. We added a bilateral grammar rules enabling to prevent self-intersection between the different object parts. This rule applies the initial deformation to the uppermost branch of the hierarchy ε_0 with the initial target branch as source. The deformation is propagated down the hierarchy only if it does not generate intersections between the sub-branches. Branches longer than a threshold are split and new branches appear at the junction between consecutive branches at the same hierarchical level, using a call to a local L-system generation. Note that our interface makes possible to *dynamically change the deformation behavior* by activating or deactivating specific rules at run time: This is used, for instance, for interpreting a subsequent free-form deformation as a radius change only in Figure (2d). Performances for this example are reported in Section 6.9.

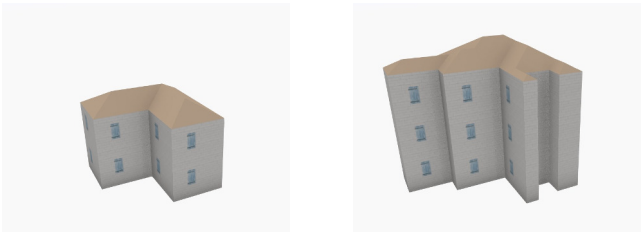


(a) (b)

Figure 6: Deformation grammars allow to freely deform organic shapes such as trees. (a): Initial tree; (b): Deformed tree. Note that the geometry of the branches is non degenerated and that junctions are evenly distributed.

6.4. Man-made objects

In order to show the versatility of the application of our deformation grammars, we set-up rules to model a deformable house. The house is a hierarchical heterogeneous object whose hierarchy is the following. A house object is composed of floors and a roof. Each floor is composed itself of walls and windows. In this example, the consistency properties are the following: adjacent walls must be orthogonal, the maximum height of each floor is bounded, and the roof is positioned at the top of the last floor. See figure 7 and 1 (second column).



(a) (b)

Figure 7: An initial house (a) is deformed by the user while preserving properties typical of man made objects such as wall orthogonality and floor linear arrangement (b).

This example also illustrates the possibility of *element specific deformation interpretation*, i.e. rules depending on the nature of the deformation: For example, a vertical translation of the roof is propagated to the house object and translated as a global vertical scaling, which allows the roof to be supported by the walls at every time. A translation of a wall piece is translated as a horizontal wall extrusion, which also results in a re-generation of the roof thanks to our bilateral grammar rules.

6.5. Object distributions

Objects distributions are hard to deform because of the inter-object constraints, such as non penetration and relative positions (see [EVC*15]). They can be represented as complex objects. Usually, the parent element in the hierarchy does not have any visual representation, but stores the distribution parameters to be maintained as internal parameters. The objects in the distribution are its children. We implemented three different examples in order to illustrate the ability of deformation grammars to maintain the consistency in the case of distributions.

The first example, shown in Figure (8), is a forest, i.e. a distribution of trees. Naively applying a user-defined deformation to this forest would either create empty regions between adjacent trees, or make some of them too close to each other. We aim at preserving the visual density of trees. This requires to merge trees that are too close, and to create new ones in large empty spaces.
Let us consider that the initial trees are associated to an underlying Delaunay triangular mesh whose vertices are the tree positions. Displacing the trees is expressed as the deformation of the mesh. Our solution for maintaining the visual appearance of the distribution is based on quasi-uniform meshes [SCC11], which are re-expressed as a specific case of our deformation grammar, a follows: Mesh vertices are maintained at a distance d such that $\frac{d_{detail}}{2} < d < d_{detail}$ (where d_{detail} is a constant learned from the input distribution). The edge collapse and split operations used to maintain the distribution's consistency trigger the elements merging and splitting respectively, which are new types of deformations.

We also show a similar example in Figure 9, where houses can

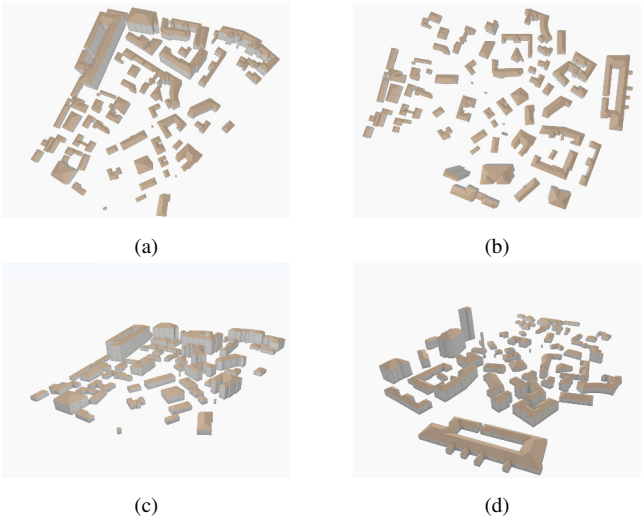
© 2017 The Author(s)
Computer Graphics Forum © 2017 The Eurographics Association and John Wiley & Sons Ltd.

(a) (b)

Figure 8: (a) Initial 2D distribution of trees. (b) Deformed distribution with newly inserted trees in the stretched regions.

split or merge based on the same rules. But this time, splitting a large house results in creating several smaller ones, while merging has the opposite effect. Such effect could be used, for instance, for compacting a village while preserving the number of inhabitants it can house.

The third example (Figure (1), 3rd column), is a volumetric distribution. Elements are merged when they come close to each other (using an element-specific merge transformation), therefore avoiding any intersection. In this case, a grid-based acceleration structure was used to compute element neighborhood.



(a) (b)

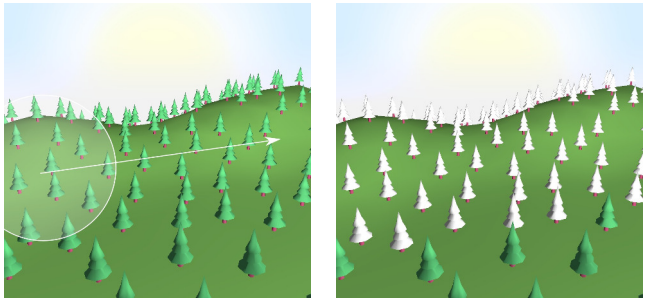
(c) (d)

Figure 10: An initial city (left column) is deformed by the user using affine and free-form deformations for creating another city (right column). Here, the city element dispatches deformations to its sub-elements, i.e. the houses presented in Section 6.4.

The last example is shown in Figure 10. It represents a city generated from GIS data of the town of Moscow. Footprints are used for generating buildings of the same type as the house of Section 6.4. Performances for this example are reported in Section 6.9.

6.6. Color transformation

Deformation grammars are not limited to geometrical transformation interpretation: As an illustration, we used the deformation grammar framework to implement a tool for painting an object distribution as shown in Figure (11).



(a) (b)

Figure 11: The user can paint an original set of trees (a) using a paining tool (in white). The change of color is interpreted by the rule such that only the foliage is affected, not the trunk (b).

6.7. Heterogeneous distributions

On of the main advantage of our deformation grammar is its ability to seamlessly handle heterogeneous distributions. Therefore, once a deformation behavior is described in our framework, it can be further reused as a sub-elements of a larger scene, and interact with other elements. For instance, outdoor scene such as the one illustrated in Figure 1, 4th column, is defined in assembling the deformation behavior of the tree distribution, bird flock, and house deformation where the root element is the entire scene. Each element can be either globally deformed, or individually while still preserving the individual and global consistency.

6.8. Persistent editing

Figure (12) illustrates persistent editing. In this case, a forest tree is locally deformed by the user (Figure 12b). Next, a global deformation is applied on every tree. If the persistent edit is not applied, the trees may be re-dispatched for maintaining tree density, therefore destroying all previous manual editing operations (Figure 12c). Instead, using our persistent editing method on a similar global deformation enables us to preserve the local aspect of the tree while still allowing it to be translated, and other trees to be deformed (see Figure 12d).

6.9. Performance

Tables 1 and 2 report performances of symbol translation for affine and free-form deformations respectively. The measures were made throughout the production of examples presented in previous Sections (6.3 and 6.5).

In both of these tables, the first row represents the number of interactors in the scene for the given deformation type (i.e. the number of elements able to interpret this deformation). The second row represents the number of deformations generated by the user during the sculpting session (i.e. the number of axiom symbols). The third row represents the number of symbol generated in total (including those of the second row, plus all the symbols generated internally by the grammar). Finally, the last row gives the average time spent for processing an axiom and all its subsequent symbols.

© 2017 The Author(s)
Computer Graphics Forum © 2017 The Eurographics Association and John Wiley & Sons Ltd.

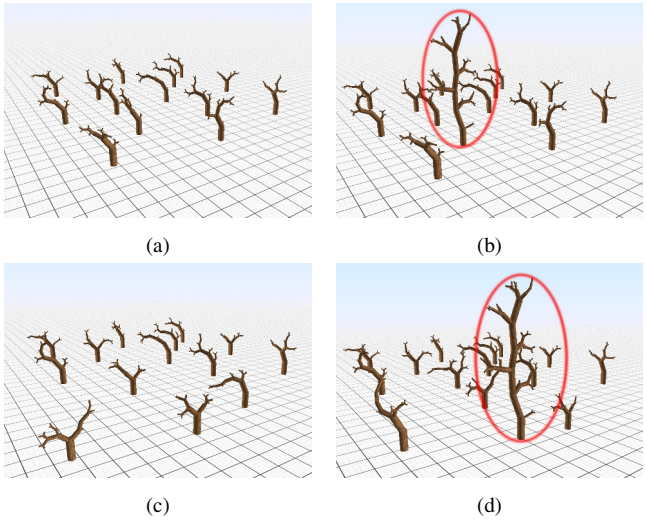


Figure 12: (a): Initial set of trees. (b): Local editing of a single tree. (c): Global deformation without persistent editing leads to a loss of the previous user edits. (d): Global deformation with our persistent editing method enables to preserve the local aspect of the previously deformed tree.

	Tree Example	City Example
# interactors	599	4740
# axioms	515	903
# symbols	125430	33223
mean interpretation time (s)	0.005	0.008

Table 1: Performance evaluations in the case of **affine** deformations interpretation.

	Tree Example	City Example
# interactors	599	4740
# axioms	31	21
# symbols	13855	1116690
mean interpretation time (s)	0.13	0.43

Table 2: Performance evaluations in the case of **free-form** deformations interpretation.

The tree example has a deep hierarchy, which explains the big number of symbol generated in total compared to the number of axioms: Each affine symbol is interpreted through the hierarchy, which creates new symbols. The hierarchy of the city example is more shallow, which explains the lower ratio.

Both examples yield interactive computation times with affine deformations, which is adequate for sculpting. On the other hand, the free-form deformations require more time, due to the deformation field evaluation and the increased complexity of the interpretation. Improvements have to be made on this side, for example by using the independence of downward symbol translation on non-hierarchically-connected elements with a parallel implementation.

User perspective. Using deformation grammars is easy: they offer a front-end interface to deformation interpretation. On the other end, creating a deformation grammar can prove to be a non trivial task. It requires to identify the design needs in terms of deformation behavior and consistency, possibly through discussion with an end-user artist; The deformation behavior has to be formalized into deformation interpretation rules; Finally those rules have to be coded into a programming language (following the process described in Section 6.2).

This whole process, although very generic, intrinsically couples the object definition with the deformation interpretation and may therefore not be non trivial to define or code. The total time to define a fully functional deformation grammar may range between some hours to days depending on the complexity of the object with respect to its hierarchy and consistency.

7. Discussion

Two methods aiming at deforming complex objects can be relevantly compared to ours in terms of results: WorldBrush [EVC*15] and TreeSketch [LRBP12]. Each of these methods focuses on a particular type of complex object: distributions of elements and trees, respectively. Although we do not provide in our implementation the specific user interfaces dedicated to trees and elements distributions, enabling to achieve the high quality interaction provided in these prior works, our deformation grammar would enable to capture similar deformation behaviors: Indeed, all the sub-elements used in these two works only require to me moved, built, and deleted while maintaining specific rules. On the one hand, integrating the histogram preservation as a consistency criteria would enable to interactively deform a 2D distribution of elements similarly to WorldBrush. On the other hand, implementing new deformation types such as element painting, and new branch behavior for trees, would enable to model the deformation behavior of TreeSketch. One of the advantages of using our deformation grammar in such cases would be to fully integrate these two different behaviors within a single framework. Then, within the same scene, the user could seamlessly design the distribution of trees of a forest, while being able to control each of the trees similarly to the approach in TreeSketch.

7.1. Advantages and drawbacks of our approach

Suitability for deforming procedurally generated objects. As seen in Section 6.3, deformation grammars are particularly well suited to interact with shapes defined using shape grammars. These objects are hierarchical by nature and they can be generated dynamically, enabling us to easily set rules that add or delete parts of the object when the latter is deformed.

Faithfulness trade-off. We call *faithfulness* of a deformation interpretation the difference between the non interpreted and the interpreted deformation.

The faithfulness is positively correlated with the predictability of the deformation behavior, and therefore to the intuitiveness of the deformation tool: The more the interpreted deformation corresponds to the input deformation (i.e. the deformation interpretation

© 2017 The Author(s)
Computer Graphics Forum © 2017 The Eurographics Association and John Wiley & Sons Ltd.

is faithful), the more the object will behave in the way the user expects it to. On the other hand, a perfectly faithful deformation interpretation will probably not respect the consistency of the object, losing all interest. A compromise is to be found between consistency preservation and interpretation faithfulness.

Over-constrained consistency. Related to the faithfulness, an object with many constrains may not provide enough degrees of freedom to be deformed as expected. As interpreted deformation will fall inside some very limited deformation space, which may result into a deformation behavior of little interest. For instance, a cube constrained to stay cubic will only allow uniform scaling deformations which may be considered too restrictive by the user.

Note, however, that even with restrictive individual behaviors, the deformation of complex heterogeneous objects combining different types of elements at different levels of a hierarchy will still look rich and expressive.

Stochasticity. Stochasticity allows grammars to choose which of several applicable rules to use according to a random law [RMGH15]. This property is heavily used with shape generation grammars for allowing various shapes to be generated from a single input. In the case of deformation grammars, the variation of the deformation behavior is not desirable because the intuitiveness of the deformation interpretation relies on its predictability. Therefore, we did not explore this track.

8. Conclusion

We presented the first general method for specifying the deformation behavior of complex hierarchical heterogeneous objects. Our method relies on the concept of deformation grammar, which is a special case of formal grammars, where symbols are deformations of elements. We showed that this method allows us to expressively deform a large variety of complex objects, from individual shapes to object assemblies and to a composite 3D scene, while maintaining specific properties of the elements composing this object as well as their hierarchical relations.

There are several avenues for future work. Inferring rules of our deformation grammar from a set of input deformed objects using inverse procedural modeling could be an interesting extension of this work. Ensuring time-continuous deformations even when new elements appear or disappear could enable deformation grammars to generate animations. Finally, complex objects could also be composed of animated elements leading to a wider variety of consistency preservation.

9. Acknowledgments

This work was funded by the advanced grant no. 291184 EXPRESSIVE from the European Research Council (ERC-2011-ADG 20110209).

Appendix A: Simple tree deformation grammar

This appendix describes the deformation grammar used in the example of Section 4.2.

Object. For this example, we consider a hierarchical tree model, as described in Figure 4:

- Internal data are representing a branch geometry, in this case we consider a cylinder;
- Hierarchical data are junction points to the branch children.

In this case, the tree is a homogeneous object, all elements are of the same type $t(\varepsilon) = branch$.

Consistency. The branch consistency is defined as follows :

- *Internal consistency:* The branch's geometry is cylindrical;
- *Hierarchical consistency:* The children's geometry starts where the branch's geometry ends.

Deformation. We consider a sculpting deformation behavior using a free-form deformation $d : x \in \mathbb{R}^3 \rightarrow d(x) \in \mathbb{R}^3$ controlled by the user's mouse displacement applying a weighted local translation in the view plane.

Grammar Rule. Defining a grammar rule boils down to define $D_{internal}$ and $D_{hierarchical}$ (see Equations (3, 4)).

Let us start with $D_{internal}$, which preserves the *internal consistency* of a branch. We call p_{start} and p_{end} the extremities of the branch geometry. In order to preserve the cylindrical geometry of the branch, $D_{internal}$ to be an affine transformation whose matrix M can be expressed as:

$$\mathrm{M} = \mathrm{T} \times \mathrm{R} \times \mathrm{S}$$

where:

- $\mathrm{T} = \mathrm{trans}\,(d(p_{start}) - p_{start})$
- $\mathrm{R} = \mathrm{rot}\left(\frac{d(p_{end})-d(p_{start})}{\|d(p_{end})-d(p_{start})\|}, \frac{p_{end}-p_{start}}{\|p_{end}-p_{start}\|}\right)$
- $\mathrm{S} = \mathrm{scale}\left(\frac{p_{end}-p_{start}}{\|p_{end}-p_{start}\|}, \frac{\|d(p_{end})-d(p_{start})\|}{\|p_{end}-p_{start}\|}\right)$

and:

- $\mathrm{trans}(x)$ is the translation of vector x;
- $\mathrm{rot}(a,b)$ is the rotation from vector a to vector b;
- $\mathrm{scale}(a,s)$ is the scaling of axis a and magnitude s

In the current case, $D_{internal}$ does not disconnect a branch from its initially connected children. However, in the general case, one can define $D_{hierarchical}$ in such a way that it re-connects a branch with its children. According to Equation 5, it only requires to define the deformation from a branch ε to its child e:

$$D^{\varepsilon,e}_{hierarchical} = \mathrm{trans}(p^{e}_{start} - p^{\varepsilon}_{end})$$

Appendix B: Bilateral tree deformation grammar

This appendix describes the bilateral deformation grammar used in the example of Section 5.4.

Here the axiom $A = (\varepsilon_2, d, \varepsilon_2, \uparrow)$ only need to go up one level in the hierarchy in order to maintain the object's consistency (the connection between branches). This is performed by R_{up}, as explained in Section 5.3. The following rules that comes into play is R_{interp}, which translates the ascending symbol $(p(\varepsilon_2), d, \varepsilon_2, \uparrow)$ into

© 2017 The Author(s)
Computer Graphics Forum © 2017 The Eurographics Association and John Wiley & Sons Ltd.

a descending symbol deforming ε_2 itself using the modified affine transformation d'. We define d' as the transformation which preserves $p^{\varepsilon_2}_{start}$ while deforming $p^{\varepsilon_2}_{end}$ as follows:

$$d' = \mathrm{T} \times \mathrm{R} \times \mathrm{S} \times \mathrm{T}^{-1}$$

where:

- $\mathrm{T} = \mathrm{trans}\left(p^{\varepsilon_2}_{start}\right)$
- $\mathrm{S} = \mathrm{scale}\left(\frac{p^{\varepsilon_2}_{end} - p^{\varepsilon_2}_{start}}{\|p^{\varepsilon_2}_{end} - p^{\varepsilon_2}_{start}\|}, \frac{\|d\, p^{\varepsilon_2}_{end} - p^{\varepsilon_2}_{start}\|}{\|p^{\varepsilon_2}_{end} - p^{\varepsilon_2}_{start}\|}\right)$
- $\mathrm{R} = \mathrm{rot}\left(p^{\varepsilon_2}_{end} - p^{\varepsilon_2}_{start}\ ,\ d\, p^{\varepsilon_2}_{end} - p^{\varepsilon_2}_{start}\right)$

The new symbol $(\varepsilon_2, d', \varepsilon_2, \downarrow)$ will in turn be translated into a terminal symbol (directly modifying the visual representation of ε_2) and a composition of non terminal symbols deforming the children of ε_2 (while maintaining the connection with the latter).

Appendix C: Implementation Details

Generative grammars implementations usually fall into one of the two following categories: *formal translation* and *recurrent function calls*. *Formal translation* allows to write grammar rules in a dedicated language (such as CGA++ [SM15] for architectural design); Rules are interpreted at run time, which allows for flexible rule editing; Each rule process a symbol from a symbol pool and generates other symbols into it, there is no recursion. *Recurrent function calls* on the other hand uses hard-coded rules inside function; Chained rule application corresponds to recurrent calls to the corresponding functions.

As stated in section 6.2, we used the second paradigm; Our choice was driven by rapidity of execution and ease of design.

Interactors A given element can be deformed using different deformation types, it will therefore implement several interpretation functions. Besides, deformation interpretation might require from an element to know which deformation types are applicable to its children. For this reasons, we introduce the *interactor* pattern.

Figure 13 shows the architecture of the deformation grammar implementation we used for creating the examples of this work. It shows the general classes used for representing elements, deformations, symbols, and grammar rules; Concrete examples are given for one deformable complex object (a tree) and two deformation types (affine and free-form).

The `Object` class represent an abstract complex object element and stores generic data: a name, a visual representation, a parent, and a list of children; The last three attributes may be empty depending on the element. Specialized classes inherit the `Object` class for defining concrete objects: For example, the `Tree` class models a tree, its `generate()` method uses an L-system for creating the mesh of the branch, and calls for the generation subbranches.

`Interactor` classes are associated to a type of symbol (i.e. a type of deformation); They act as interfaces for an object able to process the corresponding deformation. Deformation interpretation procedures (i.e. grammar rules) are coded inside `process()` method of the concrete class (inherited from the interactor). For example, instances of the `Tree` class can interpret `AffineDeformations` since `Tree` inherits from `AffineInteractor`. Applying an `AffineDeformation` to a `Tree` instance is done by calling its `process()` method inherited from `AffineInteractor` with the corresponding `AffineSymbol`.

Section 4.1 tells that a symbol contains the deformation target. In this implementation, it is not explicitly required: The target of the deformation is the object having its `process()` method called.

The *interactor* paradigm allows to implement interpretation of different deformation types inside a single element type. It also gives an easy way to query the interpretability of a deformation type by an element, by casting the element into the corresponding interactor.

Sculpting framework We used a standard sculpting software largely developed independently of our grammar framework. Elements constituting complex objects are nodes of a scene graph stored into a scene management object. User actions (mouse clicks, cursor or wheel movements, key pressed) are received by an event management object. The latter infer from the user action and the system state the desired deformation, and instantiates the corresponding symbol (using $\emptyset$ as source). It then calls the appropriate `process()` methods of the currently selected elements.

Each user action creates a grammar symbol which is fully processed before the next user action is carried. Mouse gestures must be processed at around 10 fps for the deformation to be smooth and sculpture-like. This motivates the use of recursive function calls for implementing deformation grammars: The latter are faster to execute than the more generic formal interpreter.

References

[ACWK04] ANGELIDIS A., CANI M.-P., WYVILL G., KING S.: Swirling-sweepers: Constant-volume modeling. In *Graphical Models, proc. of Pacific Graphics* (2004). 2

[AFS06] ATTENE M., FALCIDIENO B., SPAGNUOLO M.: Hierarchical mesh segmentation based on fitting primitives. *The Visual Computer* (2006). 4

[AZL12] ALHASHIM I., ZHANG H., LIU L.: Detail-replicating shape stretching. In *The Visual Computer* (2012). 2

[BBP13] BARROSO S., BESUIEVSKY G., PATOW G.: Visual copy and paste for procedurally modeled buildings by ruleset rewriting. *Computers and Graphics* (2013). 2

[BPF*22] BOUDON F., PRUSINKIEWICZ P., FEDERL P., GODIN C., KARWOWSKI R.: Interactive design of bonsai tree models. *Computer Graphics Forum, Proc. Eurographics* (22). 2

[BWKS11] BOKELOH M., WAND M., KOLTUN V., SEIDEL H.-P.: Pattern-aware shape deformation using sliding dockers. In *ACM TOG, proc. of SIGGRAPH* (2011). 2

[BWSK12] BOKELOH M., WAND M., SEIDEL H.-P., KOLTUN V.: An algebraic model for parameterized shape editing. In *ACM TOG, proc. of SIGGRAPH* (2012). 2

[Cit] CITYENGINE: Esri, http://www.esri.com/software/cityengine. 2

[DK14] DEKKERS E., KOBBELT L.: Geometry seam carving. In *Computer-Aided Design* (2014). 2

[EBP*12] EMILIEN A., BERNHARDT A., PEYTAVIE A., CANI M.-P., GALIN E.: Procedural generation of villages on arbitrary terrains. *The Visual Computer* (2012). 2, 5

© 2017 The Author(s)
Computer Graphics Forum © 2017 The Eurographics Association and John Wiley & Sons Ltd.

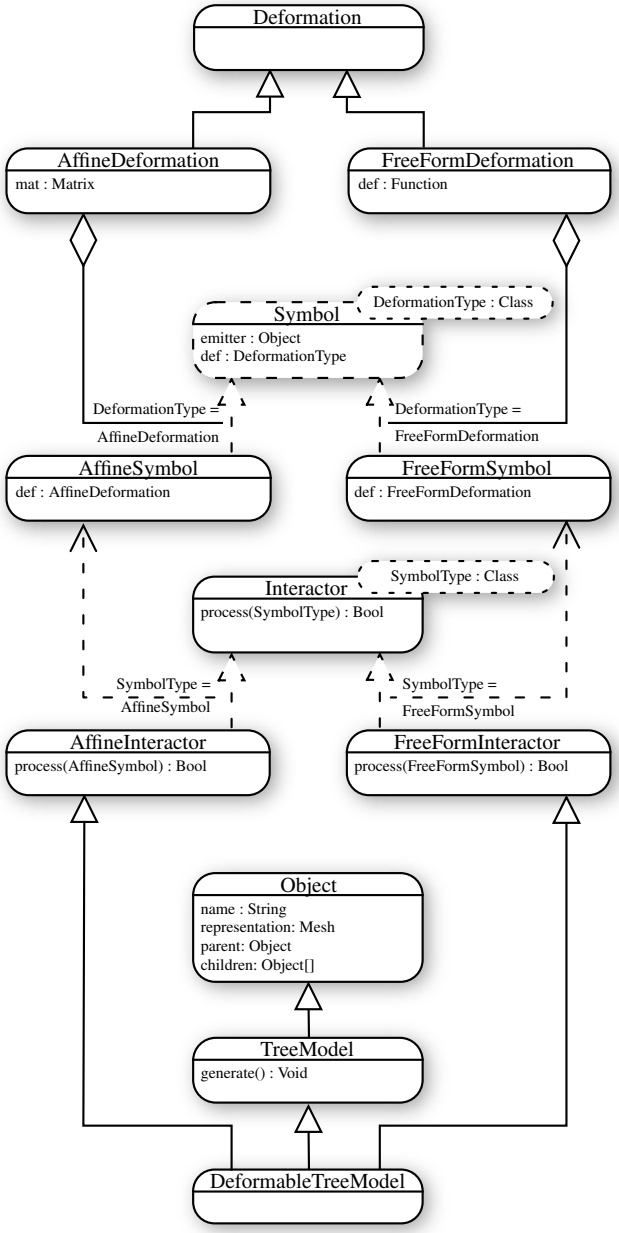


Figure 13: UML class diagram of our C++ implementation of deformation grammars through interactors. According to standard UML notations: —▷ stands for inheritance; ···≻ stands for dependency; ···▷ stands for realization; And —◇ stands for aggregation.

[EPCV14] EMILIEN A., POULIN P., CANI M.-P., VIMONT U.: Interactive procedural modelling of coherent waterfall scenes. In *CGF, proc. of Eurographics* (2014). 3

[EVC*15] EMILIEN A., VIMONT U., CANI M.-P., POULIN P., BENES B.: Worldbrush: Interactive example-based synthesis of procedural virtual worlds. In *ACM TOG, proc. of SIGGRAPH* (2015). 2, 8, 10

[FP99] FEDERL P., PRUSINKIEWICZ P.: Virtual laboratory: an interactive software environment for computer graphics. *Computer Graphics International* (1999). 2

[GPCP13] GONZÀLEZ F., PARADINAS T., COLL N., PATOW G.: *cages: : A multi-level, multi-cage based system for mesh deformation. *ACM Transactions on Graphics* (2013). 2

[GSMCO09] GAL R., SORKINE O., MITRA N. J., COHEN-OR D.: iwires: an analyze-and-edit approach to shape manipulation. In *ACM TOG, proc. of SIGGRAPH* (2009). 2

[IMAW15] ILCIK M., MUSIALSKI P., AUZINGER T., WIMMER M.: Layer-based procedural design of facades. *Computer Graphics Forum* (2015). 2

[JPCC14] JORDAO K., PETTRÉ J., CHRISTIE M., CANI M.-P.: Crowd sculpting: A space-time sculpting method for populating virtual environments. In *CGF, proc. of Eurographics* (2014). 3

[KSSCO08] KRAEVOY V., SHEFFER A., SHAMIR A., COHEN-OR D.: Non-homogeneous resizing of complex models. In *ACM TOG, proc. of SIGGRAPH* (2008). 2

[KW11] KELLY T., WONKA P.: Interactive architectural modeling with procedural extrusions. *ACM Transactions on Graphics* (2011). 2

[LRBP12] LONGAY S., RUNIONS A., BOUDON F., PRUSINKIEWICZ P.: Treesketch: interactive procedural modeling of trees on a tablet. In *proc. of the international symposium on sketch-based interfaces and modeling* (2012). 2, 3, 10

[LSWW11] LIPP M., SCHERZER D., WONKA P., WIMMER M.: Interactive modeling of city layouts using layers of procedural content. *Computer Graphics Forum* (2011). 2

[LVW*15] LIU H., VIMONT U., WAND M., CANI M.-P., HAHMANN S., ROHMER D., MITRA N. J.: Replaceable substructures for efficient part-based modeling. In *CGF, proc. of Eurographics* (2015). 2

[LWW08] LIPP M., WONKA P., WIMMER M.: Interactive visual editing of grammars for procedural architecture. In *ACM TOG, proc. of SIGGRAPH* (2008). 3

[ML13] MIAO Y., LIN H.: Visual saliency guided global and local resizing for 3d models. In *Computer-Aided Design and Computer Graphics (CAD/Graphics)* (2013). 2

[MWCS13] MILLIEZ A., WAND M., CANI M.-P., SEIDEL H.-P.: Mutable elastic models for sculpting structured shapes. In *CGF, proc. of Eurographics* (2013). 3

[MWH*06] MÜLLER P., WONKA P., HAEGLER S., ULMER A., VAN GOOL L.: Procedural modeling of buildings. In *ACM TOG, proc. of SIGGRAPH* (2006). 5

[MWZ*13] MITRA N. J., WAND M., ZHANG H., COHEN-OR D., BOKELOH M.: Structure-aware shape processing. In *Eurographics - State of the Art Reports* (2013). 2

[PBPS99] POWER J. L., BRUSH A. J. B., PRUSINKEIWICZ P., SALESIN D. H.: Interactive arrangement of botanical l-system models. *Symposium on Interactive 3D Graphics* (1999). 2

[PL12] PRUSINKIEWICZ P., LINDENMAYER A.: *The algorithmic beauty of plants*. Springer Science & Business Media, 2012. 4

[RMGH15] RITCHIE D., MILDENHALL B., GOODMAN N. D., HANRAHAN P.: Controlling procedural modeling programs with stochastically-ordered sequential monte carlo. 11

[SA07] SORKINE O., ALEXA M.: As-rigid-as-possible surface modeling. In *Symposium on Geometry Processing* (2007). 2

[SCC11] STANCULESCU L., CHAINE R., CANI M.-P.: Freestyle: Sculpting meshes with self-adaptive topology. In *Computers & Graphics* (2011). 2, 3, 8

[SCCS13] STANCULESCU L., CHAINE R., CANI M.-P., SINGH K.: Sculpting multi-dimensional nested structures. In *Computers & Graphics* (2013). 3

© 2017 The Author(s)
Computer Graphics Forum © 2017 The Eurographics Association and John Wiley & Sons Ltd.

[SKK*14] STEINBERGER M., KENZEL M., KAINZ B., MUELLER J., WONKA P., SCHMALSTIEG D.: On-the-fly generation and rendering of infinite cities on the gpu. *Computer Graphics Forum, Proc. Eurographics* (2014). 2

[SM15] SCHWARZ M., MÜLLER P.: Advanced procedural modeling of architecture. 5, 12

[vFTS06] VON FUNCK W., THEISEL H., SEIDEL H.-P.: Vector field based shape deformations. In *ACM TOG, proc. of SIGGRAPH* (2006). 2

[YCHK15] YUMER M. E., CHAUDHURI S., HODGINS J. K., KARA L. B.: Semantic shape editing using deformation handles. In *ACM TOG, proc. of SIGGRAPH* (2015). 2

[ZFCO*11] ZHENG Y., FU H., COHEN-OR D., AU O. K.-C., TAI C.-L.: Component-wise controllers for structure-preserving shape manipulation. In *CGF, proc. of Eurographics* (2011). 2

[ZLDM16] ZHENG Y., LIU H., DORSEY J., MITRA N. J.: Ergonomics-inspired reshaping and exploration of collections of models. *IEEE Transactions on Visualization and Computer Graphics* (2016). 2

© 2017 The Author(s)
Computer Graphics Forum © 2017 The Eurographics Association and John Wiley & Sons Ltd.

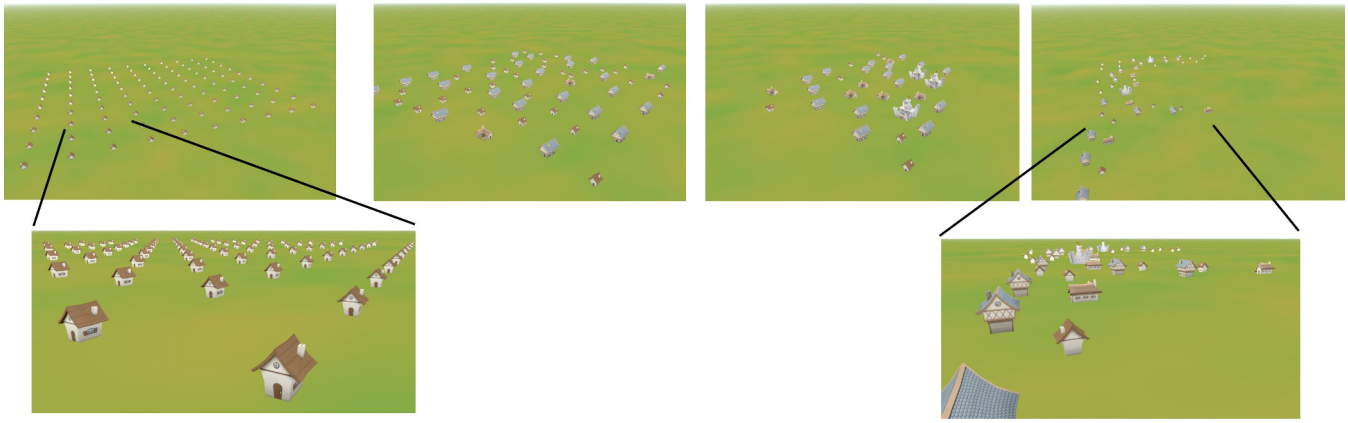


Figure 9: An initial distribution of houses (in left) is interactively deformed by the user using space deformation (middle and right figures). Closed-by houses are merged into larger ones to model the increased density.

© 2017 The Author(s)
Computer Graphics Forum © 2017 The Eurographics Association and John Wiley & Sons Ltd.